Exhibit 819-3

ENCYCLOPEDIA OF COMPUTER SCIENCE AND TECHNOLOGY

REVISED EDITION

HARRY HENDERSON

Facts On File
An imprint of Infobase Publishing

In memory of my brother,
Bruce Henderson,
who gave me my first opportunity to explore
personal computing almost 30 years ago.

ENCYCLOPEDIA OF COMPUTER SCIENCE AND TECHNOLOGY, Revised Edition

Facts On File, Inc.
An imprint of Infobase Publishing
132 West 31st Street
New York NY 10001

Library of Congress Cataloging-in-Publication Data

Henderson, Harry, 1951–
Encyclopedia of computer science and technology / Harry Henderson.—Rev. ed.
p. cm.
Includes bibliographical references and index.
ISBN-13: 978-0-8160-6382-6
ISBN-10: 0-8160-6382-6
1. Computer science—Encyclopedias. 2. Computers—Encyclopedias. I. Title.
QA76.15.H43 2008
004.03—dc22 2008029156

Facts On File books are available at special discounts when purchased in bulk quantities for businesses, associations, institutions, or sales promotions. Please call our Special Sales Department in New York at (212) 967-8800 or (800) 322-8755.

You can find Facts On File on the World Wide Web at http://www.factsonfile.com

Text design by Erika K. Arroyo
Cover design by Salvatore Luongo
Illustrations by Sholto Ainslie
Photo research by Tobi Zausner, Ph.D.

Printed in the United States of America

VB Hermitage 10 9 8 7 6 5 4 3 2 1

This book is printed on acid-free paper and contains
30 percent postconsumer recycled content.

of interest can be enhanced, and arbitrary ("false") colors can be used to visually show such things as temperature or blood flow. These techniques can also be used to create interactive models where scientists can, for example, combine molecules in new ways and have the computer calculate the likely properties of the result. Finally, computer visualization and modeling can be used both to teach science and to give the general public some visceral grasp of the meaning of scientific theories and discoveries.

Further Reading

Heath, Michael T. *Scientific Computing: An Introductory Survey.* 2nd ed. New York: McGraw-Hill, 2002.

Jähne, Bernd. *Practical Handbook of Image Processing for Scientific Applications.* 2nd ed. Boca Raton, Fla.: CRC Press, 2004.

Langtangen, H. P., A. M. Bruaset, and E. Quak, eds. *Advances in Software Tools for Scientific Computing.* New York: Springer, 2000.

Linux Software: Scientific Applications. Available online. URL: http://linux.about.com/od/softscience/Linux_Software_Scientific_Applications.htm. Accessed August 20, 2007.

Oliveira, Suely, and David E. Stewart. *Writing Scientific Software: A Guide to Good Style.* New York: Cambridge University Press, 2006.

Scientific Computing and Numerical Analysis FAQ. Available online. URL: http://www.mathcom.com/corpdir/techinfo.mdir/index.html. Accessed August 20, 2007.

scripting languages

There are several different levels at which someone can give commands to a computer. At one end, an applications programmer writes program code that ultimately results in instructions to the machine to carry out specified processing (see PROGRAMMING LANGUAGES and COMPILER). The result is an application that users can control in order to get their work done.

At the other end, the ordinary user of the application uses menus, icons, keystrokes, or other means to select program features in order to format a document, calculate a spreadsheet, create a drawing, or perform some other task. Today most users also control the operating system by using a graphical user interface to, for example, copy files.

However, there is an intermediate realm where text commands can be used to work with features of the operating system, to process data through various utility programs, and to create simple reports. For example, a system administrator may want to log the number of users on the system at various times, the amount of disk capacity being used, the number of hits on various pages on a Web server, and so on. (See SYSTEM ADMINISTRATOR.) It would be expensive and time-consuming to write and compile full-fledged application programs for such tasks, particularly if changing needs will dictate frequent changes in the processing.

The use of the operating system shell and shell scripting (see SHELL) has traditionally been the way to deal with automating routine tasks, especially with systems running UNIX. However, the complexity of modern networks and in particular, the Internet, has driven administrators and programmers to seek languages that would combine the quick, interactive nature of shells, the structural features of full-fledged programming languages, and the convenience of built-in facilities for pattern-matching, text processing, data extraction, and other tasks. The result has been the development of a number of popular scripting languages (see AWK, PERL, and PYTHON).

Working with Scripting Languages

Although the various scripting languages differ in syntax and features, they are all intended to be used in a similar way. Unlike languages such as C++, scripting languages are interpreted, not compiled (see INTERPRETER). Typically, a script consists of a number of lines of text in a file. When the file is invoked (such as by someone typing the name of the language followed by the name of the script file at the command prompt), the script language processor parses each statement (see PARSER). If the statement includes a reference to one of the language's internal features (such as an arithmetic operator or a print command), the appropriate function is carried out. Most languages include the basic types of control structures (see BRANCHING STATEMENTS and LOOP) to test various variables and direct execution accordingly.

The trend in higher-level languages has been to require that all variables be declared to be used for particular kinds of data such as integer, floating-point number, or character string (see DATA TYPES). Scripting languages, however, are designed to be easy to use and scripts are relatively simple and easy to debug. Since the consequences of errors involving data types are less likely to be severe, scripting languages don't require that variables be declared before they are used. The language processor will make "common sense" assumptions about data. Thus if an integer such as 23 and a floating-point number like 17.5 must be added together, the integer will be converted to floating point and the result will be expressed as the floating-point value 40.5.

Similarly, scripting languages take a relaxed view about scope, or the parts of a program from which a variable's value can be accessed. Scripting languages do provide for some form of subroutine or procedure to be declared (see PROCEDURES AND FUNCTIONS). Generally, variables used within a subroutine will be considered to be "local" to that subroutine, and variables declared outside of any subroutine will be treated as global.

With compilers for regular programming languages, a great deal of attention must be paid to creating fast, efficient code. A scientific program may need to optimize calculations so that it can tackle cutting-edge problems in physics or engineering. A commercial application such as a word processor must implement many features to be competitive, and yet be able to respond immediately to the user and complete tasks quickly.

Scripting languages, on the other hand, are typically used to perform housekeeping tasks that don't place much demand on the processor, and that often don't need to be finished quickly. Because of this, the relative inefficiency of on-the-fly interpretation instead of optimized compilation is not a problem. Indeed, by making it easy for users to write and test programs quickly, the interpreter makes it much easier for administrators and others to create simple but

useful tools for monitoring the system and extracting necessary information. Scripting languages can also be used to quickly create a prototype version of a program that will be later recoded in a language such as C++ for efficiency.

Scripting languages were originally written for operating systems that process text commands. However, with the popularity of Microsoft Windows, Macintosh, and various UNIX-based graphical user interfaces, many users and even system administrators now prefer a visual scripting environment. For example, Microsoft Visual Basic for Windows (and the related Visual Basic for Applications and VBScript) allow users to write simple programs that can harness the features of the Windows operating system and user interface and take advantage of prepackaged functionality available in ActiveX controls (see BASIC). In these visual environments the tasks that had been performed by script files can be automated by setting up and linking appropriate objects and adding code as necessary.

Web Scripting

Aside from shell programming, the most common use of scripting languages today is to provide interactive features for Web pages and to tie forms and displays to data sources. On the Web server, such technologies as ASP (see ACTIVE SERVER PAGES) use scripts embedded in (or called from) the HTML code of the page. On the client side (i.e., the user's Web browser) languages such as JavaScript and VBScript can be used to add features.

(For specific scripting languages, see AWK, JAVASCRIPT, LUA, VBSCRIPT, PERL, PHP, and TCL. For general-purpose languages that have some features in common with scripting languages, see PYTHON and RUBY.)

Further Reading

Barron, David. *The World of Scripting Languages*. New York: Wiley, 2000.

Brown, Christopher. "Scripting Languages." Available online. URL: http://cbbrowne.com/info/scripting.html. Accessed August 20, 2007.

Foster-Johnson, Eric, John C. Welch, and Micah Anderson. *Beginning Shell Scripting*. Indianapolis: Wiley, 2005.

Ousterhout, John K. "Scripting: Higher Level Programming for the 21st Century." *IEEE Computer*, March 1998. Available online. URL: http://www.tcl.tk/doc/scripting.html. Accessed August 20, 2007.

"Scriptorama: A Slightly Skeptical View on Scripting Languages." Available online. URL: http://www.softpanorama.org/Scripting/index.shtm. Accessed August 20, 2007.

Sebesta, Robert W. *Concepts of Programming Languages*. 8th ed. Boston: Pearson/Addison-Wesley, 2007.

search engine

By the mid-1990s, many thousands of pages were being added to the World Wide Web each day (see WORLD WIDE WEB). The availability of graphical browsing programs such as Mosaic, Netscape, and Microsoft Internet Explorer (see WEB BROWSER) made it easy for ordinary PC users to view Web pages and to navigate from one page to another. However, people who wanted to use the Web for any sort of systematic research found they needed better tools for finding the desired information.

There are basically three approaches to exploring the Web: casual "surfing," portals, and search engines. A user might find (or hear about) an interesting Web page devoted to a business or other organization or perhaps a particular topic. The page includes a number of featured links to other pages. The user can follow any of those links to reach other pages that might be relevant. Those pages are likely to have other interesting links that can be followed, and so on. Most Web users have surfed in this way: It can be fun and it can certainly lead to "finds" that can be bookmarked for later reference. However, this approach is not systematic, comprehensive, or efficient.

Alternatively, the user can visit a site such as the famous Yahoo! started by Jerry Yang and David Filo (see PORTAL and YAHOO!). These sites specialize in selecting what their editors believe to be the best and most useful sites for each topic, and organizing them into a multilevel topical index. The portal approach has several advantages: The work of sifting through the Web has already been done, the index is easy to use, and the sites featured are likely to be of good quality. However, even Yahoo!'s busy staff can examine only a tiny portion of the estimated 1 trillion or so Web pages being presented on about 175 million different Web sites (as of 2008). Also, the sites selected and featured by portals are subject both to editorial discretion (or bias) and in some cases to commercial interest.

Anatomy of a Search Engine

Search engines such as Lycos and AltaVista were introduced at about the same time as portals. Although there is some variation, all search engines follow the same basic approach. On the host computer the search engine runs automatic Web searching programs (sometimes called "spiders" or "Web crawlers"). These programs systematically visit Web sites and follow the links to other sites and so on through many layers. Usually, several such programs are run simultaneously, from different starting points or using different approaches in an attempt to cover as much of the Web as possible. When a Web crawler reaches a site, it records the address (URL) and compiles a list of significant words. The Web crawlers give the results of their searches to the search engine's indexing program, which adds the URLs to the associated keywords, compiling a very large word index to the Web.

Search engines can also receive information directly from Web sites. It is possible for page designers to add a special HTML "metatag" that includes keywords for use by search engines. However, this facility can be misused by some commercial sites to add popular words that are not actually relevant to the site, in the hope of attracting more hits.

To use a search engine, the user simply navigates to the search engine's home page with his or her Web browser. (Many browsers can also add selected search engines to a special "search pane" or menu item for easier access.) The user then types in a word or phrase. Most search engines accept logical specifiers (see BOOLEAN OPERATORS) such as